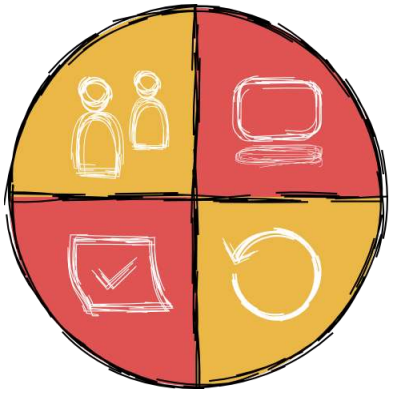
# Green patches in your brownfield

# about me

- programmer + DevOps enthusiast
  - Java, Groovy, learning Clojure
  - Linux, Ansible, Liquibase, Maven
- co-organizing software craftsmanship groups:
  - Ruhrgebiet + Düsseldorf (Germany)
  - coderetreats, coding dojos





georgberky


georg.berky@valtech-mobility.com

# terminology

- brownfield
  - undeveloped land
  - muddy, not nice to walk on
  - existing codebase, untested code
- greenfield
  - new project
  - nice to walk on
  - test drive your code from the start

# food for thought


# today’s agenda

- some theory + lots of live coding
- seams, enabling points, types of refactoring
- sprouting
- wrap class
- split loop

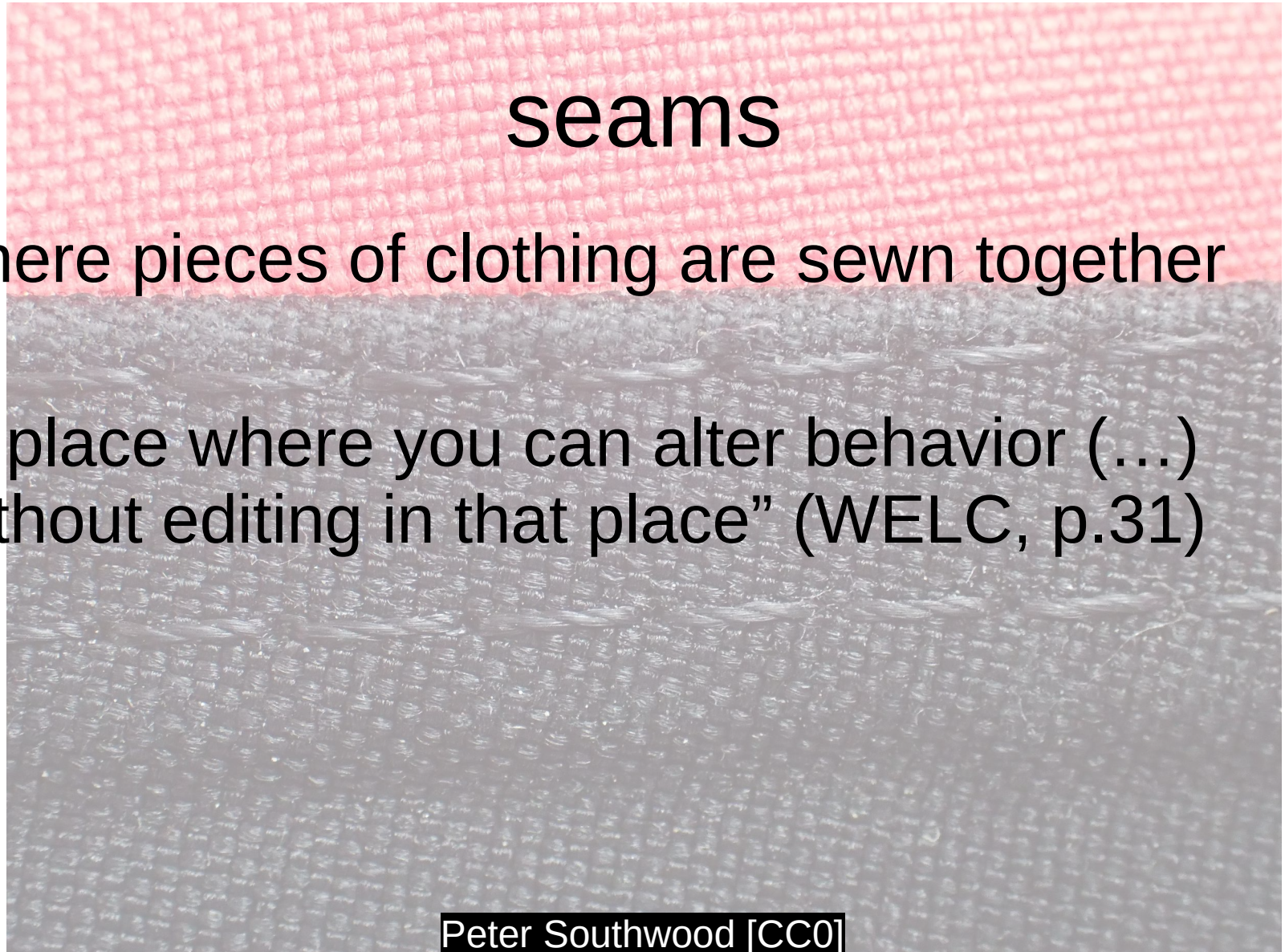
# sources

- WELC:
  Michael C. Feathers – Working effectively with Legacy Code, Prentice Hall, 2004
- REF:
  Martin Fowler – Refactoring (2nd edition), Boston : Addison-Wesley, 2019

# seams

- where pieces of clothing are sewn together

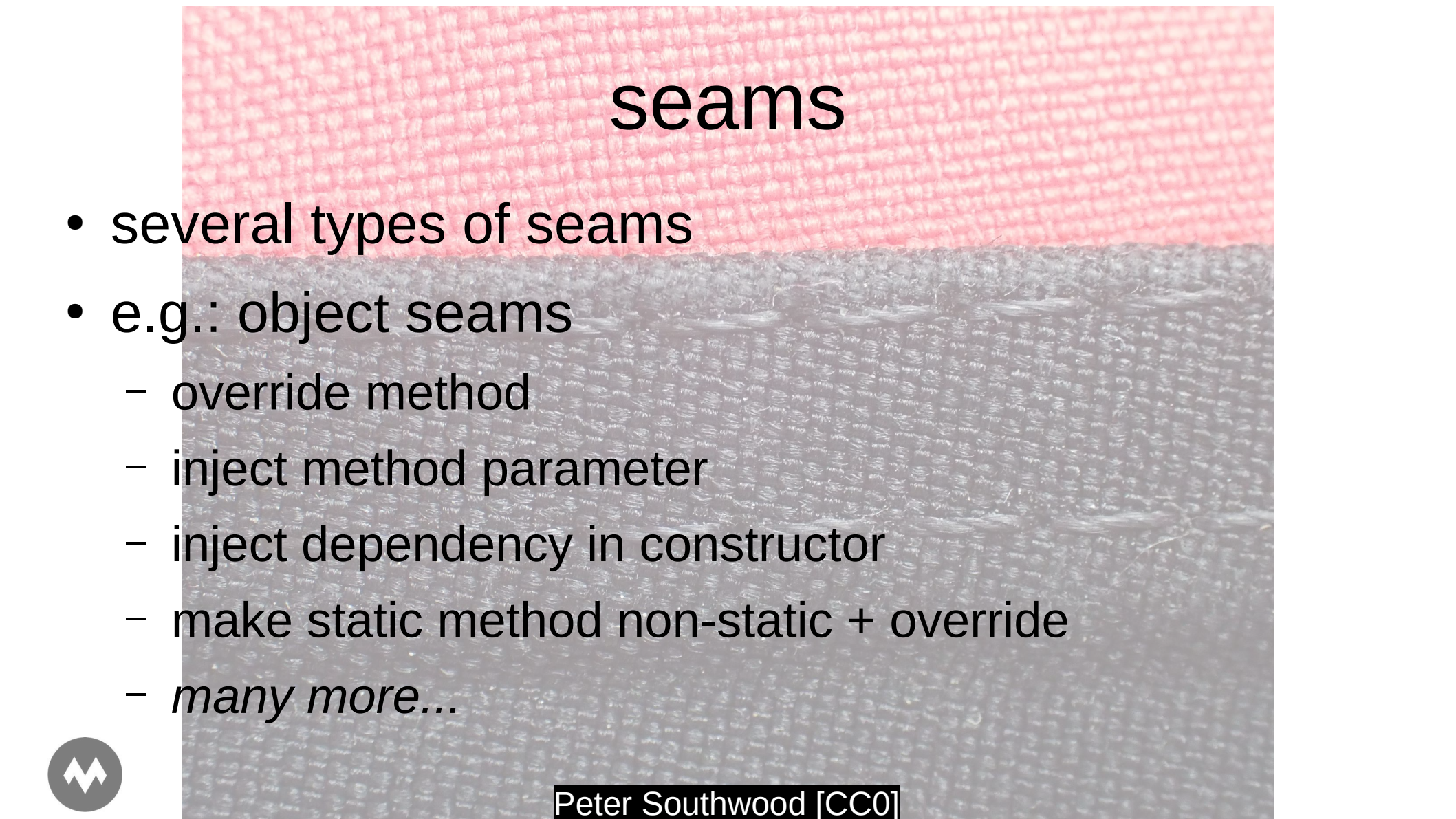
- ”a place where you can alter behavior (…) without editing in that place” (WELC, p.31)



Peter Southwood [CC0]

# seams

- several types of seams
- e.g.: object seams
  - override method
  - inject method parameter
  - inject dependency in constructor
  - make static method non-static + override
  - *many more...*



Peter Southwood [CC0]

# enabling points

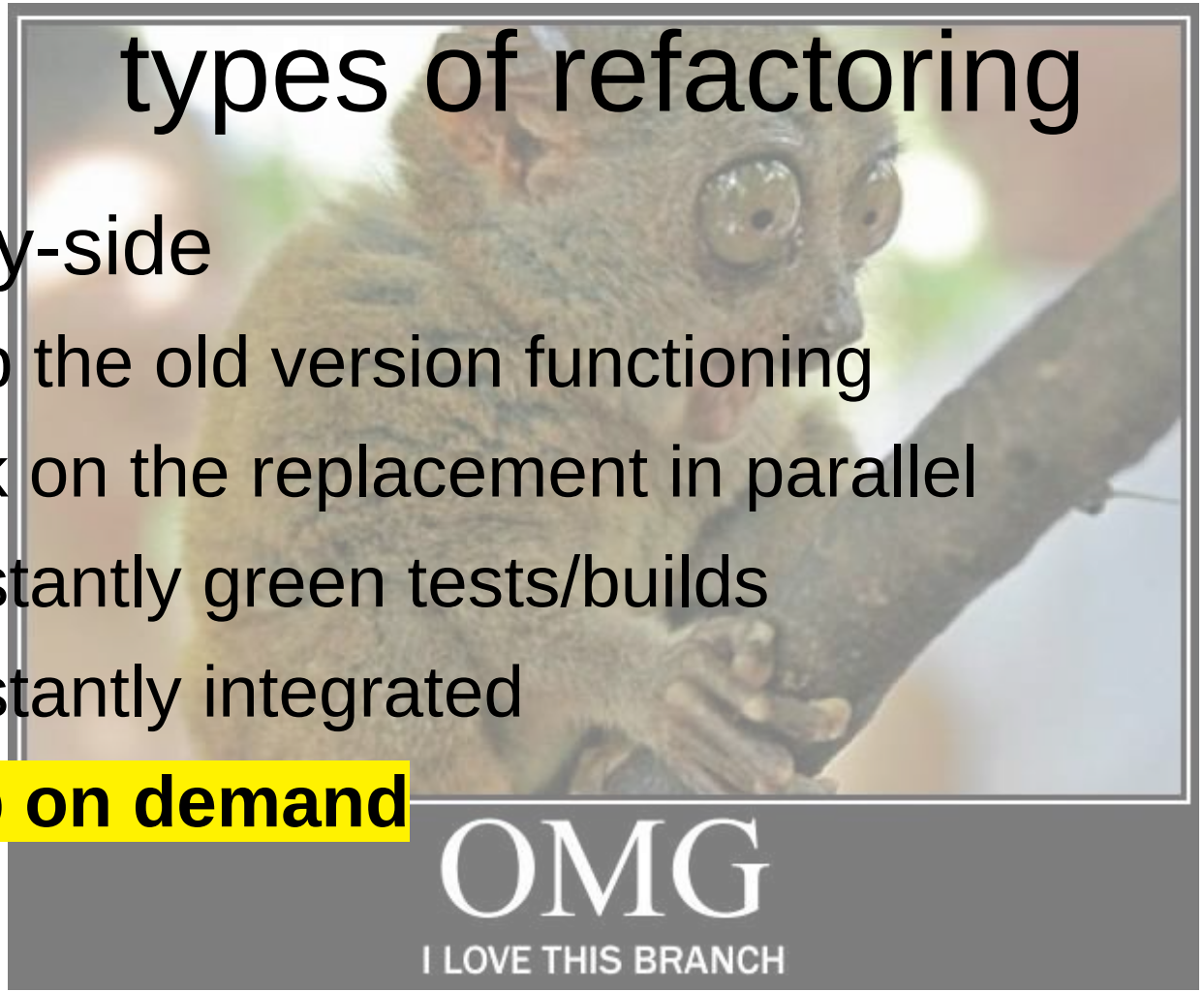
- ”Every seam has an enabling point, a place where you can make the decision to use one behavior or another” (WELC, p.36)
- object seam:
  - override in class definition
  - argument list of constructor/method

# types of refactoring

- in-situ
  - change everything on the spot
  - forces changes to other spots
  - tests/builds stay red longer
  - isolated: branch/local commits
  - merge before being able to ship
  - **cannot ship on demand**

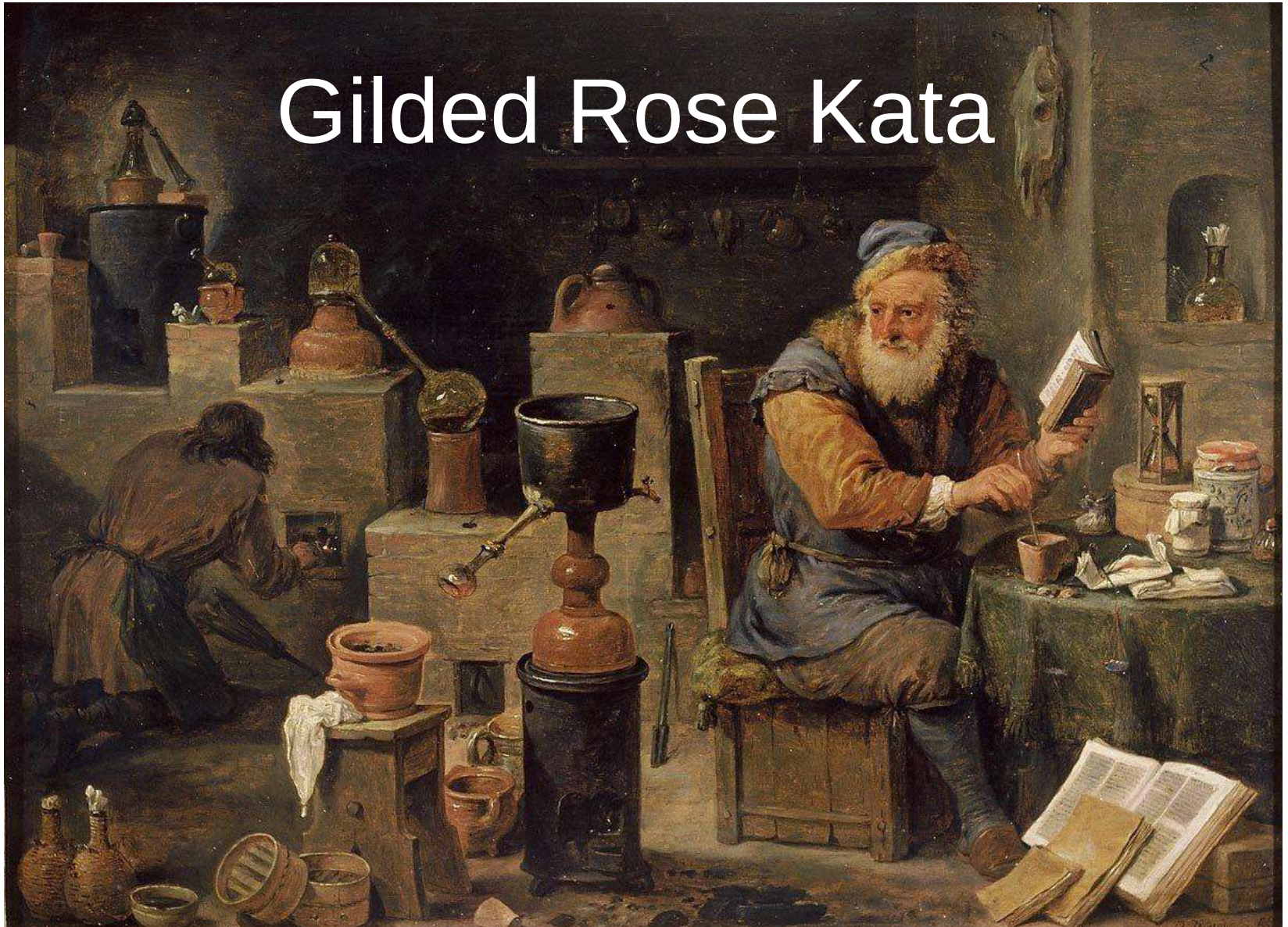
# types of refactoring

- side-by-side
  - keep the old version functioning
  - work on the replacement in parallel
  - constantly green tests/builds
  - constantly integrated
  - **ship on demand**




GeePaw Hill - Refactoring: Side-by-Side v. In Situ

# Gilded Rose Kata



# preparations

- add tests before you refactor
  - characterization tests
  - use dependency breakers from WELC
  - golden master
- safety net for refactoring

# sprouting

- **when:** you don’t have much time
- new functionality in new method/class
- test-drive all new code
- call from existing code
- long term:
  - similarities between sprouts
  - refactor to new design

WELC, p. 59ff

# wrap class

- **when:** you cannot change a class
  - 3rd party library
  - goblin in the corner
- wrap the class + gold plate it
- antidote for: anemic data model

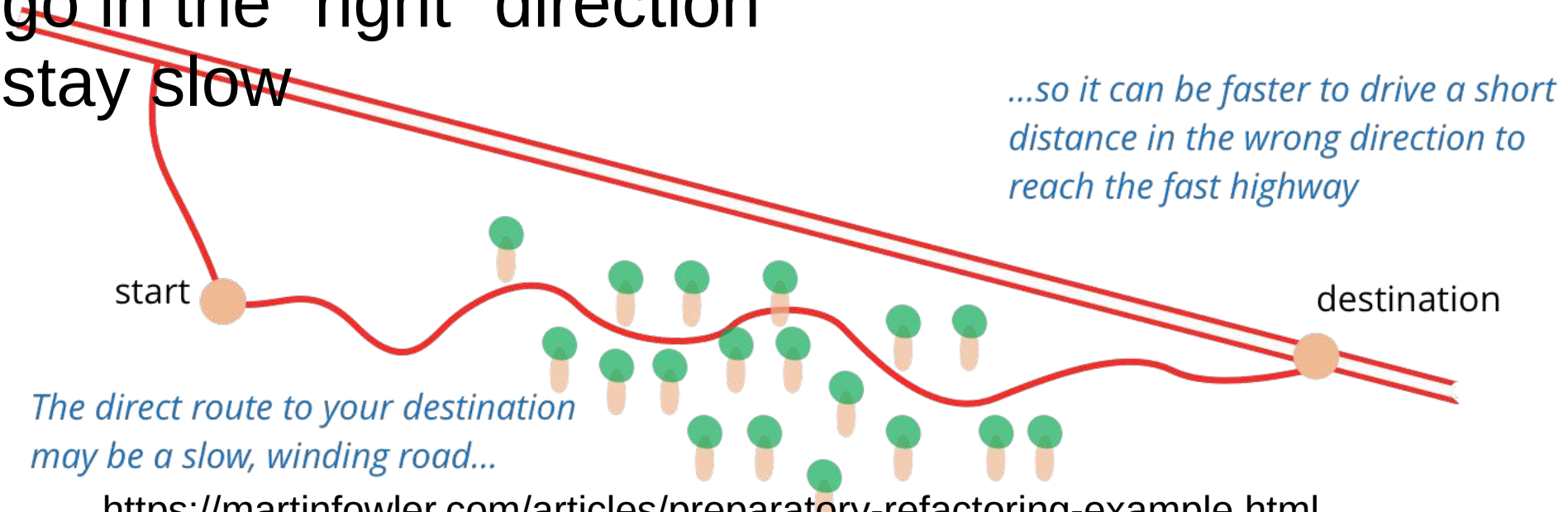
Martin Howard (Flickr, CC BY-SA 2.0)

# wrap class

- gold-plate Item
- create wrapper for Item: GildedItem
- use wrapper in production code
- intermediate mess: aliasing
  - Items in GildedRose
  - Items wrapped by GildedItem

# intermediate “mess”

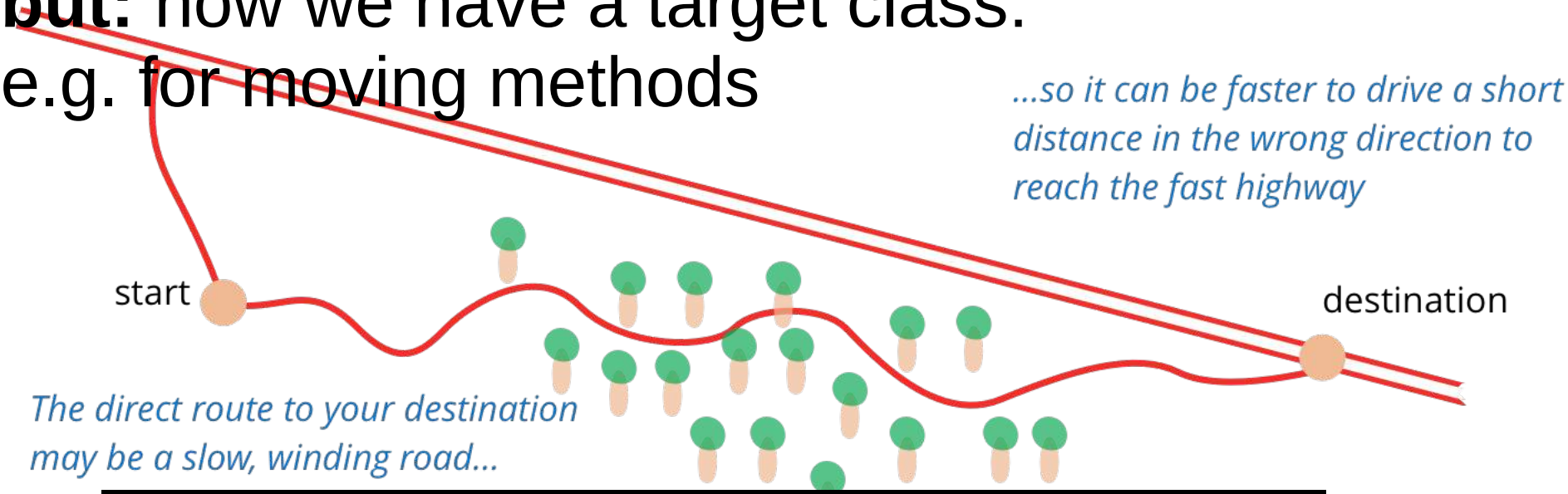
- reach the highway
  by going in the “wrong” direction first
- go in the “right” direction
  stay slow




https://martinfowler.com/articles/preparatory-refactoring-example.html

# intermediate “mess”

- “wrong” direction first:
  Item and GildedItem side-by-side
- **but:** now we have a target class:
  e.g. for moving methods




https://martinfowler.com/articles/preparatory-refactoring-example.html

# split loop

- **when:** multiple computations tangled in one loop
- copy the loop
- identify and eliminate other computations
  - use test coverage markers
- clean up if possible:
  - slide statements
  - extract function
- test

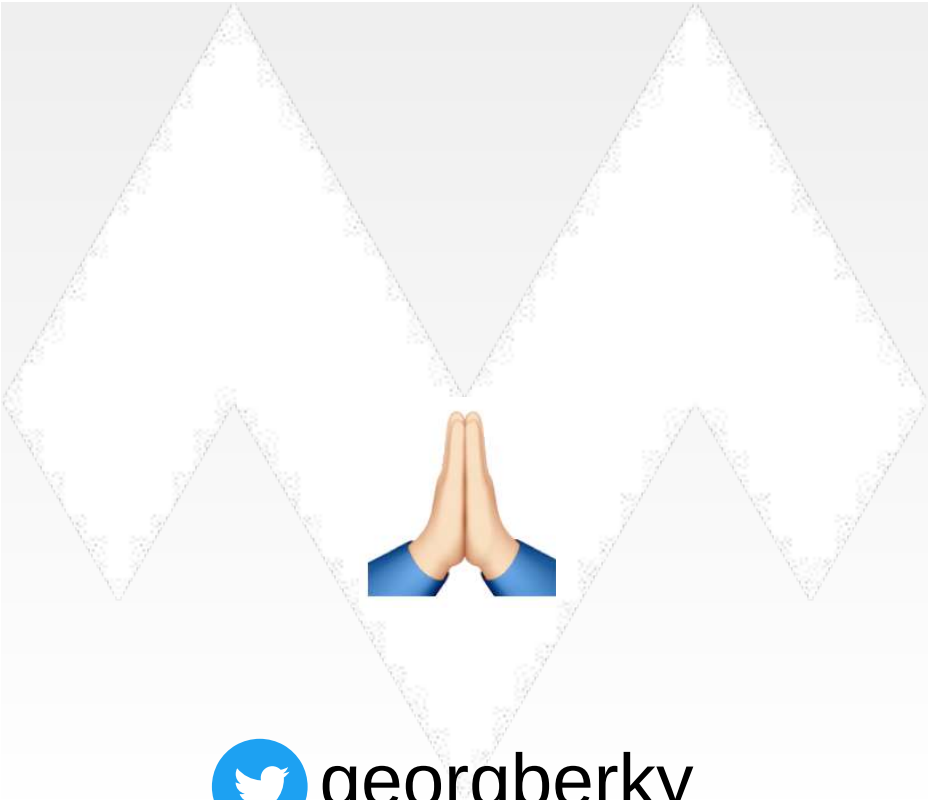
REF, p. 227ff

Arbitrarily0 (CC BY-SA 3.0)

# sources

- WELC:
  Michael C. Feathers – Working effectively with Legacy Code, Prentice Hall, 2004
- REF:
  Martin Fowler – Refactoring (2nd edition), Boston : Addison-Wesley, 2019

# thank you!



georgberky

georg.berky@valtech-mobility.com